برمجة قواعد بيانات SQLITE

في لازاروس



إعداد

م. أبوبكر شرف الدين سويدان

2015

الفهرس

مقدمة وتمهيد

لازاروس Lazarus

هي بيئة تطويرية متكاملة، متعددة المنصات، حرة ومجانية، مفتوحة المصدر. تستخدم مترجم فري باسكال الذي يدعم أوبجكت باسكال. وإن سألك أحدهم بم تطور برامجك، قل له: أستخدم لغة فري باسكال – بيئة لازاروس.



يمكن لمطوري سطح المكتب استخدام فري باسكال – بيئة لازاروس لتطوير تطبيقات تعمل على عدة أنظمة تشغيل مثل وندوز ولينوكس وماك.

نتاج البرمجة في فري باسكال – بيئة لازاروس هي ملفات تنفيذية طبيعية Native بأحجام صغيرة نسبياً، سريعة التنفيذ، خفيفة على الذاكرة.

تشبه بيئة لازاروس إلى حد كبير بيئة دلفي لتطوير التطبيقات، حيث توفر مجموعة كبيرة من الأدوات والعناصر المساعدة على برمجة التطبيقات بسهولة بالغة.

توفر بيئة لازاروس عناصر تساعد على إنشاء واجهة المستخدم بحيث ما تراه هو ما تحصل عليه، ومحرر للكود، والتنسيق، ومتتبع الأخطاء، وإدارة المشاريع.

كما توفر إمكانية الاتصال والتعامل مع مختلف أنواع قواعد البيانات مثل MySql وPostgresql وSqlite3 وFireBird وOracle وغيرها.

العمل على هذه البيئة سهل وممتع، ولا يحتاج منك إلا الإلمام بأساسيات لغة فري باسكال، التي لها مستندات ومجتمع زاخر بالمعلومات.

من الأمور الممتعة في فري باسكال – بيئة لازاروس، أنه يمكنك برمجة مشروع تطبيق على وندوز مثلاً، وتنتج به برنامجاً يعمل على أي جهاز به نظام وندوز بمجرد نسخ الملف التنفيذي (وبعض الملفات الضرورية للعمل مثل ملف قاعدة البيانات أو ملفات التقارير) وتشغيله دون أية مشاكل تذكر.

كما يمكنك نسخ الكود البرمجي للمشروع ونقله إلى منصة أخرى مثل لينوكس أوبونتو، وعمل Compile فينتج تطبيقاً يعمل على لينوكس دون تغيير حقيقي في الكود! هذا هو معنى الشعار Write once, Compile Anywhere.



لتنزيل نسختك من البيئة، ادخل موقع البيئة : www.lazarus-ide.org

وللمزيد من المعلومات عن لغة فري باسكال، يمكنك مطالعة الموقع الخاص بها: www.freepascal.org

وأنصح المبتدئ بتنزيل ودراسة كتب الأستاذ معتز عبد العظيم من موقعه على الإنترنت: www.code-sd.com



في هذا الكتاب سنتعلم كيفية التعامل مع قواعد بيانات SQLite3 في فري باسكال – بيئة لازاروس، على نظام وندوز من مايكروسوفت، باعتبار أن الغالبية العظمي من المستخدمين العرب يستعملون وندوز، بحيث نستكشف المكونات التي صممت لتسهيل العمل، ونتعرف على خصائصها ووظائفها وأحداثها.

سنعمل على إنشاء مشروع لتطبيق خاص بدليل هاتف مبسط، نجري عليه كل العمليات، من إضافة وبحث وتعديل وحذف، كما سنتعلم بعض الأمور البرمجية التي سنحتاجها في سياق العمل، والتي لا علاقة لها بقواعد البيانات.

قلتُ قبل قليل أننا سنتعلم كيفية التعامل مع قواعد بيانات SQLite3 من خلال فري باسكال – بيئة لازاروس، فما هي SQLite3 أو سيكويلايت؟

قواعد بيانات SQLite3

بشكل عام، هناك نوعان من قواعد البيانات، قواعد بيانات تعتمد في عملها على خادم Server، وقواعد بيانات عبارة عن ملف واحد غير معتمد على سيرفر.

من أمثلة قواعد البيانات التي تعتمد على سيرفرات: MySql وMariaDB وPostgresql وOracle. ومن أمثلة قواعد البيانات التي لا تعتم على سيرفرات Access وSQLite3.

في البرامج الكبيرة، وحتى المواقع الكبيرة، يفضل استخدام قواعد البيانات التي تعتمد على سيرفرات لإدارتها، الأمر طبيعي، نظراً للكمية الهائلة من البيانات التي يتم إدارتها.

أما في البرامج الصغيرة، والمتوسطة، والتي نعلم – مسبقاً – أن البيانات مهما كثرت لن تصل إلى حجم معين ليحتاج إلى قواعد بيانات كبيرة جداً، فالأنسب هو استعمال النوع الثاني من قواعد البيانات، وأقصد تلك التي تكون على هيئة ملف واحد.

ولكي نستطيع العمل مع قواعد بيانات سيكويلايت، نحتاج لوجود ملف مكتبة خاص بها (ذو امتداد dll.)، هذه المكتبة كُتبت بلغة C، تعتبر محركاً لقواعد البيانات SQL مضمناً وصغير الحجم (لا يتعدى حجمه 700 كيلو بايت.)، يوضع في نفس مجلد التطبيق، ويتم إرفاقه مع التطبيق ليعمل على جهاز الزبون.

لهذا النوع من قواعد البيانات ميزات وعيوب، وتتمثل الميزات في كون ملف قاعدة البيانات صغير الحجم نسبياً، سهل الإنشاء والتركيب، يمكن نقل البيانات من جهاز إلى آخر بسهولة (نسخ – لصق)، لا مشاكل في الترميز وخاصة مع اللغة العربية، لا حاجة لإعدادات خاصة.

يمكن أن يتحمل ملف قاعدة البيانات حجماً يصل إلى 2048 جيجابايت أي 2 تيرابايت من البيانات كحد أقصى! جميل أليس كذلك؟

كما يمكن استعمالها في المواقع التي لا تدعم MySql.

القاعدة الرئيسية تقول: إن سيكويلايت صممت لتكون قاعدة بيانات مضمنة ضمن البرامج والتطبيقات الصغيرة والمتوسطة، لذا من الخطأ استعمالها في حالة المشاريع الكبيرة والتي تعتمد على المشاركة في العمليات بين المستخدمين. (وهذا من أهم عيوب هذا النوع من قاعدة البيانات.)

يمكن تنزيل ملف المكتبة من موقع سيكويلايت الرسمي www.sqlite.org

في ركن التنزيلات اختر sqlite-dll-win32-x86-3080803.zip، تحت البند Precompiled Binaries for Windows.

فك الضغط، وخذ نسخة من الملف sqlite3.dll وضعها في مجلد خاص لترجع إليه لاحقاً.

الباب الأول

الجزء النظري

SQLDB

SQLDB

يتم التعامل مع قواعد البيانات بكل أنواعها في فري باسكال – بيئة لازاروس من خلال (وحدة) خاصة بهذا العمل، وهي الوحدة SQLdb.

وهي عبارة عن مجموعة من الفئات (الكلاسات Classes) نستغلها للاتصال بقاعدة البيانات وإجراء مختلف العمليات عليها من إضافة سجلات وبحث وعرض وتعديل وحذف.

وبصورة أدق، تحتوي هذه الوحدة على ثلاث كلاسات أساسية للربط بين التطبيق وقاعدة البيانات تمهيداً للتعامل معها وهي:

الكلاس	الوظيفة
TSQLConnection	ويمثل اتصالاً بين التطبيق وقاعدة البيانات. ولديه مجموعة من الخصائص التي يجب ضبطها قبل الشروع في الاتصال بقاعدة البيانات وقت التشغيل. ولكل نوع من قواعد البيانات Connection خاص به، ويجب تذكّر ذلك
TSQLQuery	هو بمثابة صورة من الفئة Dataset، ونستغله في إجراء العمليات مثل استرجاع البيانات، أو إضافة السجلات، أو تعديلها وحتى حذفها بعد ربطه بالكلاس Connection
TSQLTransaction	هذا الكلاس هو المسئول عن تطبيق التغييرات التي نقوم بها على قاعدة البيانات من عدمه. بحيث يتم تنفيذ كتلة كاملة من الأوامر أو التراجع عنها دفعة واحدة. ويتم ربطه أيضاً بالكلاس Connection

هذه الكلاسات وغيرها متوفرة كأدوات يمكن استعمالها مرئياً، من خلال فتح التبويب SQLdb في أعلى الشاشة الرئيسية للبيئة، كما بالصورة التالية:



وبمجرد إدراج أحدها، يتم تضمين هذه الوحدة في البرنامج. ولكن قبل توضيح ذلك، أود أن أشرح المزيد عن هذه الكلاسات.

1 - الكلاس TSQLConnection

يمثل اتصالاً بين التطبيق الذي نبرمجه وقواعد البيانات، وبما أنه يوجد لدينا عدة أنواع من قواعد البيانات، فلكل نوع كلاس TSQLConnection خاص به.

الكلاس TSQLConnection لديه مجموعة من الخصائص التي يجب تعيين قيمها قبل البدء في الاتصال بقاعدة البيانات.

ولكن باعتبار أن قواعد البيانات يمكن تصنيفها إلى قسمين، قسم يعمل من خلال سيرفر، وقسم قائم بذاته، فإن الخصائص ستختلف في الحالتين.

فمثلاً، في حالة الاتصال بقاعدة بيانات MySql، هذا النوع يعتمد على سيرفر، وبالتالي سنختار الـConnection الملائم لها من التبويب SQLdb، وهو أحد الإصدارات التالية:



وتعني الأرقام في الصورة إصدارات كل سيرفر، وبالتالي سيعتمد اختيارك للـConnection على رقم إصدارة MySql لديك.

وفي هذه الحالة، سيتم تعيين قيم مجموعة من الخصائص قبل الاتصال وهي: HostName وDatabaseName وUserName وPassword.

أما في حالة النوع الثاني من قواعد البيانات وهو النوع القائم بذاته على هيئة ملف واحد، فخصائص الـConnection المطلوب تعيين قيمها تختلف، فلا نحتاج إلى HostName ولا إلى UserName ولا Password.

فعلي سبيل المثال، عند الاتصال بقاعدة بيانات SQLite3 – وهو موضوع الكتاب – سنعين قيمة الخاصية DatabaseName فقط، وبكل تأكيد سيختلف نوع الـConnection عن النوع سالف الذكر.

وعلى كل حال، سأشرح كافة الخصائص آنفة الذكر، بشيء من التفصيل، حتى يستفيد من يريد الاتصال بأنواع أخرى من قواعد البيانات:

الخاصية	الوظيفة
HostName	وتمثل اسم السيرفر المراد الاتصال به، أو رقم الـIP الخاص به، قد يكون هذا السيرفر محلياً (على نفس كمبيوتر الزبون) أو كمبيوتر ضمن شبكة محلية، أو قد يكون مستضافاً على الإنترنت
DatabaseName	وتمثل اسم قاعدة البيانات التي نريد الاتصال بها وفتحها وإجراء العمليات عليها
UserName	وتمثل اسم المستخدم المسموح له بالاتصال بسيرفر قاعدة البيانات ولديه صلاحيات معينة
Password	وتمثل كلمة المرور الخاصة بالمستخدم للتحقق من أهليته للاتصال بسيرفر قاعدة البيانات

هذه الخصائص كما قلت هي خاصة بالـConnection الذي سيتصل بسيرفر قاعدة بيانات، ولا نحتاجها للاتصال بقاعدة بيانات SQLite3 مثلا.

هذه هي الخصائص الأساسية التي يجب تحديد قيمها قبل محاولة الاتصال بسيرفر قاعدة البيانات، أي إهمال لإحدى هذه الخصائص يولد خطأ، ولن يتم الاتصال بنجاح، ولكن توجد خصائص أخرى متممة لها، منها:

الخاصية	الوظيفة
CharSet	ومن خلالها يمكن ضبط نوع الترميز المستخدم في قاعدة البيانات المطلوبة. بالنسبة لي، أفضل دوماً utf8
Connected	وهي خاصية منطقية، ترجع قيمة True إذا نجح الاتصال بقاعدة البيانات، وFalse إذا فشل الاتصال. وتستخدم للتحقق من اتصال التطبيق بقاعدة البيانات

كما يدعم الـConnection مجموعة من الإجراءات أو الأوامر، منها:

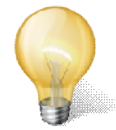
الإجراء	الوظيفة
ExecuteDirect	ويستخدم لتطبيق أوامر SQL على قاعدة البيانات. كعمليات إنشاء الجداول وإدخال البيانات
GetTableNames	ويستخدم لاسترجاع قائمة بأسماء الجداول الموجودة في قاعدة البيانات المحددة
GetFieldNames	ويستخدم لاسترجاع قائمة بأسماء الحقول الموجودة في الجدول المحدد
CreateDB	ويستخدم لإنشاء قاعدة بيانات جديدة. (يعمل مع قواعد البيانات التي تعتمد على سيرفر)
DropDB	ويستخدم لحذف قاعدة بيانات موجودة. (يعمل مع قواعد البيانات التي تعتمد على سيرفر)
Open	ويستخدم لفتح الاتصال
Close	ويستخدم لإغلاق الاتصال
Free	ويستخدم لتحرير الموارد المستعملة من قبل الـConnection

صورة أيقونة الكلاس Connection الخاص بقواعد بيانات SQLite3 هي:



البرمجة في فري باسكال – بيئة لازاروس مسيرة بالأحداث. بمعنى أنه لن يتم تنفيذ الكود إلا عند وقوع حدث ما، كنقر زر، أو تحميل Form، أو عند إغلاقه، أو عند النقر المزدوج على كائن ما، أو انتقال التركيز Focus، أو غير ذلك.

أنسب حدث لتعيين قيم خصائص الـConnection هو الحدث OnCreate التابع للفورم الرئيسي للتطبيق. بينما يكون الحدث OnActivate لنفس الفورم هو الحدث الأنسب للبدء في الاتصال بقاعدة البيانات.



يجب الانتباه إلى نقطة مهمة في حالة قواعد بيانات SQLite3:

إذا قمنا بتحديد اسم ملف قاعدة البيانات للـConnection، ولا يوجد الملف فعلياً، سيتم إنشاء ملف جديد لقاعدة بيانات فارغة (لا تحتوي على جداول أو بيانات) بالاسم المعطى.

ولذلك، يجب التأكد من وجود ملف قاعدة البيانات قبل البدء في تشغيل التطبيق فعلياً، والقيام بالخطوات الضرورية في حالة عدم وجود الملف.

قلت في ما سبق، أنه عندما نتعامل مع قاعدة البيانات SQLite3 ، فإننا نحتاج إلى مكتبة خاصة، أقصد تحديداً الملف (sqlite3.dll) الذي سبق وأنزلناه من الموقع الرسمي لـSQLite3 .

ولكي نستعمل هذا الملف في التطبيق، نستخدم الكلاس TSQLDBLibraryLoader.

الكلاس TSQLDBLibraryLoader



هذا الكلاس هو من ضمن كلاسات الوحدة SQLDB، ووظيفته تحميل المكتبات (dll,so,dylibs) وتعريفها للتطبيق.

يدعم هذا الكلاس مجموعة من الخصائص وهي:

الخاصية	الوظيفة
ConnectionType	وتمثل نوع قاعدة البيانات المرغوب الاتصال بها. في حالتنا ستكون SQLite3
LibraryName	وتمثل مسار واسم المكتبة، في حالتنا ستكون sqlite3.dll
Enabled	خاصية منطقية، True تعني أن الكلاس يعمل
LoadLibrary	ونستخدمها للبدء في تحميل المكتبة إلى الذاكرة

أنسب حدث لتحديد قيم هذه الخصائص هو الحدث OnCreate للفورم الرئيسي للتطبيق.

2 – الكلاس TSQLQuery

هو بمثابة صورة من الفئة Dataset، ونستغله لإجراء العمليات على البيانات في قاعدة البيانات، كالبحث والاسترجاع والعرض، وإضافة سجلات، وتعديلها، وحذفها بعد ربطه بالكلاس Connection.

من أهم خصائص هذا الكلاس:

الخاصية	الوظيفة
Database	وهي خاصية نصية، وتكون مساوية لاسم الـConnection.
SQL.Text	وهي المكان الذي نكتب فيه جمل SQL لتنفيذها على قاعدة البيانات.
ParamByName	ونستخدمها لتعيين قيم البارامترات في جملة SQL المطلوب تنفيذها في SQL.Text

كما للكلاس TSQLQuery إجراءات منها:

الإجراء	الوظيفة
Open	ونستخدمه للبدء في تنفيذ استعلامات SQL المحددة، والتي تكون لغرض البحث واسترجاع البيانات فقط.
ExecSQL	ونستخدمه للبدء في تنفيذ استعلامات SQL المحددة، والتي تكون خاصة بعمليات الإضافة والتعديل والحذف.

بعبارة أخرى، نستخدم الإجراء Open في حالة جملة select، ونستخدم الإجراء ExecSQL في حالات insert وupdate وdelete.

صورة أيقونة هذا الكلاس هي:



الكلاس TDatasource

في حالة استعمال الإجراء Open أي أن العملية هي عملية بحث واسترجاع للسجلات، ستكون نتيجة الاستعلام هي Dataset تتكون من سجل أو أكثر، سنحتاج إلى كائن يستقبل الـDataset المسترجعة ومن ثم يمكننا استغلاله لعرض البيانات المخزنة في تلك الـDataset.

هذا الكلاس هو TDatasource، وهو لا يتبع الوحدة SQLDB، بل يتبع الوحدة DB.

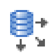
يمكننا وضع هذا الكلاس على الفورم - لاستعماله برمجياً - من خلال لسان التبويب Data Access في لازاروس، وهو عنصر غير مرئي وقت التشغيل. يتم ربطه بالكلاس TSQLQuery بعد إجراء عملية البحث والاسترجاع من خلال الخاصية Dataset التابعة له.

من أهم خصائص هذا الكلاس:

الخاصية	الوظيفة
Dataset	تحدد الـ TSQLQuery الذي تسترجع منه السجلات، وتكون مساوية لاسم الـTSQLQuery

بعد ذلك يتم ربط العناصر التي نرغب في عرض البيانات المسترجعة عليها مثل صناديق النصوص Edits وLabels وTDBGrids وغيرها عن طريق الخاصية Datasource التي تتبعها.

صورة أيقونة هذا الكلاس هي:



3 – الكلاس TSQLTransaction

وهو المسئول عن تنفيذ أو التراجع عن العمليات التي يقوم بها الكلاس TSQLQuery .

من أهم إجراءات (أوامر) هذا الكلاس:

الإجراء	الوظيفة
Commit	ويقوم بتأكيد العملية التي قام بها الكلاس TSQLQuery.
RollBack	ويقوم بالتراجع عن العملية الأخيرة.
StartTransaction	ويقوم بالتمهيد للعملية الجديدة.
EndTransaction	ويقوم بإنهاء العملية الجديدة.

يتم ربط هذا الكلاس بالكلاس TSQLConnection من خلال الخاصية Transaction التي تتبع الأخير.

صورة أيقونة هذا الكلاس هي:



وخلاصة القول، أن أي عملية إضافة أو تعديل أو حذف لن تتم فعلياً إلا من خلال هذا الكلاس.

الخلاصة

لكي نستطيع التعامل مع قواعد البيانات SQLite3 في لازاروس نستخدم الكلاسات التالية:

- الكلاس TSQLDBLibraryLoader لتحميل مكتبة sqlite3.dll.
- الكلاس TSQLConnection للاتصال بقاعدة البيانات أو إنشائها.
- الكلاس TSQLQuery لإجراء العمليات على قواعد البيانات.
- الكلاس TSQLTransaction لإتمام أو التراجع عن العمليات.
- الكلاس TDatasource لاستقبال السجلات المسترجعة وتوزيعها على عناصر العرض.

الباب الثاني

الجزء العملي

برنامج إدارة جهات الاتصال

فكرة عامة

برنامج إدارة جهات الاتصال هو برنامج بسيط، يمكن لمستخدمه تسجيل بيانات الأشخاص والجهات التي يتصل بها، وتعديلها وحذفها، وحتى إظهار تقرير وسحبه على الورق.

أي أن البيانات التي سيسجلها المستخدم هي:

- اسم جهة الاتصال
- أرقام الهواتف
- البريد الإلكتروني

إذن، سنصمم جدولاً لتخزين هذه البيانات، وسيكون بالصورة التالية:

Contacts		
الحقل	الوصف	النوع
Id	يمثل الرقم التسلسلي للسجل	INTEGER
contactName	يمثل اسم جهة الاتصال	STRING
phoneNumbers	يمثل أرقام الهواتف	STRING
emailAddress	يمثل البريد الإلكتروني	STRING

تنويه

أحب أن أشير إلى أن قواعد بيانات SQLite3 لا تدعم خاصية الزيادة التلقائية في حقول ID، ولذا سنعمل حركة التفاف عليها عن طريق إنشاء جدول خاص (بالعدادات) بالصورة التالية:

Counters		
الحقل	الوصف	النوع
Id	يمثل الرقم التسلسلي للسجل، وسيكون دائماً = 1	INTEGER
contactid	سيمثل عداد الـID الخاص بالجدول Contacts	INTEGER

ما يعني أنه عند إنشاء الجدول Counters برمجياً، يتوجب علينا إدراج سجل بالصورة التالية:

Counters	
الحقل	القيمة
Id	1
contactid	0

يعني:

- نعطي أمر إنشاء الجدول Counters وحقوله.
- ندرج سجلاً جديداً.

ولاحقاً، عند إدراج سجل جديد في الجدول Counters ، نقوم بالآتي:

1. نبحث في الجدول Counters عن قيمة الحقل contactid ، ونخزنها في متغير وليكن اسمه LastID.
2. نضيف إليه (1)، يعني ;LastID := LastID + 1.
3. نقوم بتعديل الحقل contactid في الجدول Counters بحيث يأخذ القيمة الجديدة.
4. الآن نستخدم المتغير LastID لنخزنه في الجدول Contacts في الحقل ID.
5. نخزن باقي الحقول في نفس الصف.

وينطبق ما قيل هنا على إضافة سجل جديد لأي جدول آخر.

ربما يكون الكلام مبهماً الآن، ولكن سنفهمه عند التطبيق العملي إن شاء اللّه.

خطة العمل

- إنشاء واجهة المستخدم.
- التأكد من وجود ملف قاعدة البيانات:
  - في حالة عدم وجوده:
    - برمجة إجراء إنشاء قاعدة البيانات.
  - في حالة وجوده:
    - الاتصال بقاعدة البيانات.
- إضافة السجلات.
- البحث عن السجلات.
- تعديل / حذف السجلات.
- إنشاء التقرير.

إلى العمل ...

إنشاء واجهة المستخدم User Interface

- نفتح مشروعاً جديداً في لازاروس.
- نسمي النافذة الأولى بالاسم: frmMain، والوحدة الخاصة به باسم frmMainUnit.
- نعدل خصائص النافذة كما في الجدول التالي:

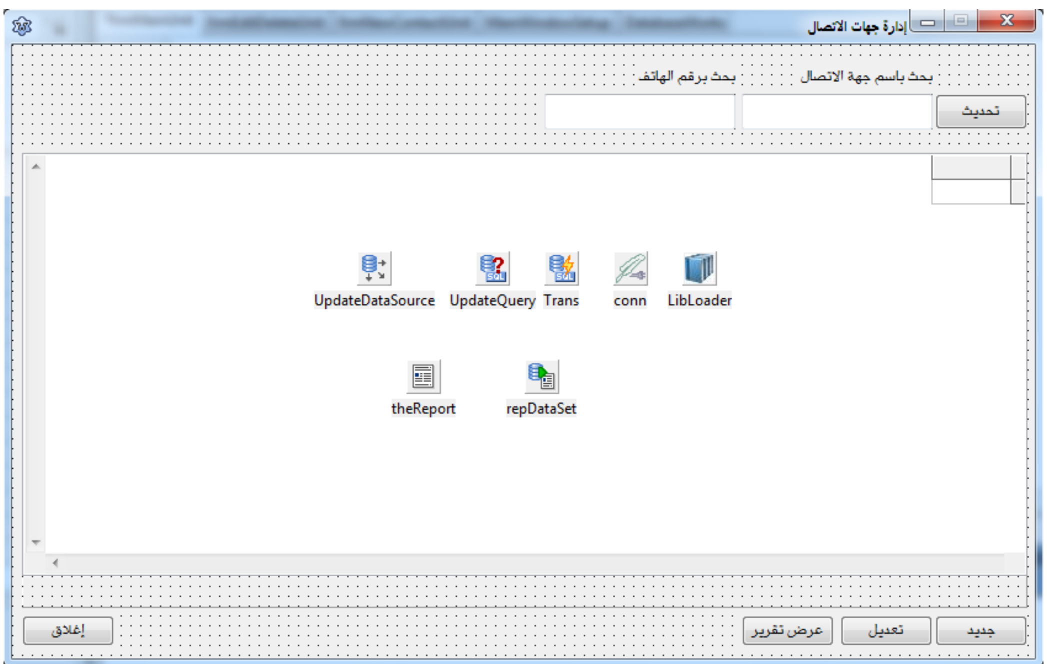
الخاصية	القيمة
BiDiMode	bdRightToLeft
BorderIcons -> biMaximize	False
BorderStyle	bsSingle
Caption	إدارة جهات الاتصال
Position	poScreenCenter

- نخزن المشروع باسم Contacts في مجلد خاص على سطح المكتب.
- نضع العناصر التالية على النافذة frmMain، ونعدل الخصائص بالصورة التالية:

العنصر	الخاصية	القيمة
TDBGrid	BiDiMode	bdRightToLeft
	Options-> dgAlwaysShowSelection	False
	Options-> dgRowSelect	True
	ScrollBars	ssVertical
	Name	dgvContacts
TButton	Name	btnNew
	Caption	جديد
TButton	Name	btnEdit
	Caption	تعديل
	Enabled	False
TButton	Name	btnShowReport
	Caption	عرض تقرير
TButton	Name	btnClose
	Caption	إغلاق
TButton	Name	btnRefresh

	Caption	تحديث
TEdit	Name	txtSearchInNames
TEdit	Name	txtSearchInNumbers
TPanel	Name	pnlSelectedContact
TSQLDBLibraryLoader	Name	LibLoader
TSQLite3Connection	Name	conn
TSQLTransaction	Name	Trans
TSQLQuery	Name	UpdateQuery
UpdateDataSource	Name	UpdateDataSource
TfrReport	Name	theReport
TfrDBDataSet	Name	repDataSet

وننسقها حتى يصير الشكل العام كما في الصورة التالية:



التأكد من وجود ملف قاعدة البيانات

1- تمهيد

هل تتذكر – عزيزي – ملف sqlite3.dll الذي أنزلناه من موقع قاعدة البيانات SQLite3؟ فقد حان وقت تحميله وربطه بالتطبيق.

- في البداية، ننسخ ذلك الملف ونضعه في مجلد المشروع.
- نعود إلى لازاروس، نختار الفورم frmMain، ونذهب إلى نافذة الخصائص.
- في التبويب Events ننقر على الحدث OnCreate نقرتين سريعتين فيتم إظهار كود الحدث Create للفورم frmMain كما بالصورة التالية:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin

end;
```

نكتب الكود التالي بين Begin وEnd:

```
LibLoader.ConnectionType:='SQLite3';
LibLoader.LibraryName:='sqlite3.dll';
LibLoader.Enabled:=True;
```

فيصير بالصورة التالية:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
  LibLoader.ConnectionType:='SQLite3';
  LibLoader.LibraryName:='sqlite3.dll';
  LibLoader.Enabled:=True;
end;
```

وفي هذا الكود:

- تم ضبط الخاصية ConnectionType للـLibLoader بحيث خصصناها للـSQLite3، يعني حددنا نوع قاعدة البيانات التي نريد التعامل معها في برنامجنا.
- تم ضبط الخاصية LibraryName باسم ومسار ملف محرك قاعدة البيانات وهو الملف sqlite3.dll.
- تم تفعيل الـLibLoader والبدء في تحميل المكتبة من خلال الخاصية Enabled.

2- الوحدة الخاصة بإنشاء قاعدة البيانات

قلتُ قبل قليل أنه عندما لا نجد ملف قاعدة البيانات، يتوجب علينا القيام بإجراء إنشاء قاعدة البيانات وجداولها، وبالتالي سنكتب الكود الخاص بذلك ونخزنه في وحدة خاصة، ثم نستدعي إجراء إنشاء قاعدة البيانات في الوقت والمكان المناسبين.

- في القائمة File في بيئة لازاروس نختار New Unit.
- يفتح محرر الكود وحدة برمجية جديدة باسم Unit1 كما بالصورة التالية:

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

implementation

end.
```

- نخزن هذه الوحدة في مجلد المشروع باسم: DataBaseWorks.
- تحت القسم implementation نكتب:

```
Uses frmMainUnit;
```

وهذا السطر هو للربط بين الوحدة DataBaseWorks والوحدة frmMainUnit باتجاه واحد، بحيث يمكن في الوحدة DataBaseWorks الاستفادة من العناصر الموجودة على الفورم frmUnit، وسنستغل هذه العلاقة في إنشاء قاعدة البيانات وجداولها من خلال التعامل مع كل من:

- مكون الاتصال Conn.
- المكون Trans.

وذلك من خلال إجراء procedure سنكتبه الآن في الوحدة DataBaseWorks باسم CreateDataBase بالطريقة التالية:

- تحت السطر uses frmMainUnit ننزل قليلاً (قبل السطر End) ونكتب:

```
procedure CreateDataBase;
begin

end;
```

- الآن، نعرف متغيراً يعمل في نطاق هذا الإجراء باسم QueryString وبالنوع String، قبل كلمة Begin وبالشكل التالي:

```
procedure CreateDataBase;
var
   QueryString: String;
begin

end;
```

- الآن، سنقوم بضبط بعض الخصائص المتعلقة بكائن الاتصال Conn التابع بدوره للفورم frmMain بالصورة التالية:

```
procedure CreateDataBase;
var
   QueryString: String;
begin
      frmMain.conn.DatabaseName:='Contacts.db';
      frmMain.conn.Transaction:=frmMain.Trans;
      frmMain.conn.Open;
end;
```

وفي هذا الكود:

- تم ضبط قيمة الخاصية DatabaseName للـConn باسم وامتداد ملف قاعدة البيانات، وعند تنفيذ هذا السطر سيتم إنشاء ملف قاعدة البيانات Contacts.db في نفس مجلد البرنامج، ولا يحتوي على أية جداول.
- تم ضبط مدير العمليات Trans للخاصية Transaction التابعة للـConn، وهو مهم ولا يمكن تطبيق أوامر إنشاء الجداول إلا من خلال مدير العمليات Trans.
- تم فتح اتصال بملف قاعدة البيانات من خلال الإجراء Open التابع للـConn.

الآن نعود إلى ما تحت السطر

```
Classes, SysUtils;
```

في الوحدة DataBaseWorks، ننزل قليلاً ثم ننسخ السطر:

```
procedure CreateDataBase;
```

ونلصقه تحتها فيصير الشكل العام:

```
unit DataBaseWorks;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

procedure CreateDataBase;


implementation
Uses frmMainUnit;


procedure CreateDataBase;
var
   QueryString: String;
begin
     frmMain.conn.DatabaseName:='Contacts.db';
     frmMain.conn.Transaction:=frmMain.Trans;
     frmMain.conn.Open;
end;


end.
```

والآن، نشرع في كتابة أوامر إنشاء الجداول... نبدأ على بركة اللّه...

إنشاء الجدول Counters

قلتُ بأن جدول العدادات يتكون من الحقول التالية:

Counters		
الحقل	الوصف	النوع
Id	يمثل الرقم التسلسلي للسجل، وسيكون دائماً = 1	INTEGER
contactid	سيمثل عداد الـID الخاص بالجدول Contacts	INTEGER

تحت آخر سطر كتبناه في الإجراء CreateDataBase نكتب:

```
QueryString:='CREATE TABLE IF NOT EXISTS Counters('+
                             'id INTEGER, ' +
                             'contactID INTEGER)';

frmMain.Trans.StartTransaction;

frmMain.conn.ExecuteDirect(QueryString);

frmMain.Trans.Commit;
```

وفي الكود الأخير:

- تم كتابة جملة SQL لإنشاء الجدول Counters وحقوله ، ومن ثم وضعها في المتغير QueryString لتنفيذها لاحقاً.
- إعطاء الأمر للبدء في العملية من خلال الإجراء StartTransaction التابع للـConn.
- تنفيذ الاستعلام QueryString من خلال الإجراء ExecuteDirect التابع للـConn.
- تأكيد أمر إنشاء الجدول من خلال الإجراء Commit التابع للـTrans.

بعد ذلك، نقوم بإنشاء فهرس لهذا الجدول بالطريقة التالية:

```
QueryString:='CREATE UNIQUE INDEX CountersIndex ON Counters(id ASC)';

frmMain.Trans.StartTransaction;
frmMain.conn.ExecuteDirect(QueryString);
frmMain.Trans.Commit;
```

في الكود السابق:

- تم كتابة جملة إنشاء فهرس باسم CountersIndex اعتماداً على الحقل id في الجدول Counters وتخصيصها للمتغير QueryString لتنفيذها لاحقاً.
- إعطاء الأمر للبدء في العملية من خلال الإجراء StartTransaction التابع للـConn.
- تنفيذ الاستعلام QueryString من خلال الإجراء ExecuteDirect التابع للـConn.
- تأكيد أمر إنشاء الفهرس من خلال الإجراء Commit التابع للـTrans.

بعد ذلك، نقوم بإدراج سجل جديد في الجدول Counters بالصورة التالية:

```
QueryString:='INSERT INTO Counters VALUES(1,0)';

frmMain.Trans.StartTransaction;
frmMain.conn.ExecuteDirect(QueryString);
frmMain.Trans.Commit;
```

في الكود السابق:

- تم كتابة جملة إضافة سجل جديد في الجدول Counters وتخصيصها للمتغير QueryString لتنفيذها لاحقاً.
- إعطاء الأمر للبدء في العملية من خلال الإجراء StartTransaction التابع للـConn.
- تنفيذ الاستعلام QueryString من خلال الإجراء ExecuteDirect التابع للـConn.
- تأكيد أمر إضافة السجل من خلال الإجراء Commit التابع للـTrans.

إنشاء الجدول Contacts

للتذكير، تركيبة هذا الجدول بالصورة التالية:

Contacts		
الحقل	الوصف	النوع
Id	يمثل الرقم التسلسلي للسجل	INTEGER
contactName	يمثل اسم جهة الاتصال	STRING
phoneNumbers	يمثل أرقام الهواتف	STRING
emailAddress	يمثل البريد الإلكتروني	STRING

تحت آخر سطر كتبناه في الإجراء CreateDataBase نكتب:

```
QueryString:= 'CREATE TABLE IF NOT EXISTS Contacts(' +
                         'id INTEGER,' +
                         'contactName VARCHAR(255),' +
                         'phoneNumbers VARCHAR(255),' +
                         'emailAddress VARCHAR(255))';
frmMain.Trans.StartTransaction;
frmMain.conn.ExecuteDirect(QueryString);
frmMain.Trans.Commit;
```

في هذا الكود:

- تم كتابة جملة SQL لإنشاء الجدول Contacts وحقوله ، ومن ثم وضعها في المتغير QueryString لتنفيذها لاحقاً.
- استخدمنا النوع VARCHAR لأنه أنسب للبيانات التي سيحملها الحقل Description.
- إعطاء الأمر للبدء في العملية من خلال الإجراء StartTransaction التابع لـConn.
- تنفيذ الاستعلام QueryString من خلال الإجراء ExecuteDirect التابع لـConn.
- تأكيد أمر إنشاء الجدول من خلال الإجراء Commit التابع لـTrans.

بعد ذلك، ننشئ الفهرس الخاص بهذا الجدول:

```
QueryString:='CREATE UNIQUE INDEX ContactsIndex' +
                        ' ON Contacts(id ASC)';
frmMain.Trans.StartTransaction;
frmMain.conn.ExecuteDirect(QueryString);
frmMain.Trans.Commit;
```

وفي هذا الكود:

- تم كتابة جملة إنشاء فهرس باسم ContactsIndex اعتماداً على الحقل id في الجدول Contacts وتخصيصها للمتغير QueryString لتنفيذها لاحقاً.
- إعطاء الأمر للبدء في العملية من خلال الإجراء StartTransaction التابع للـConn.
- تنفيذ الاستعلام QueryString من خلال الإجراء ExecuteDirect التابع للـConn.
- تأكيد أمر إنشاء الفهرس من خلال الإجراء Commit التابع للـTrans.

فيصير الكود كاملاً كما يلي:

```
procedure CreateDataBase;
var
   QueryString: String;
begin
      frmMain.conn.DatabaseName:='Contacts.db';
      frmMain.conn.Transaction:=frmMain.Trans;
      frmMain.conn.Open;

      QueryString:='CREATE TABLE IF NOT EXISTS Counters('+
                              'id INTEGER, ' +
                              'contactID INTEGER)';
      frmMain.Trans.StartTransaction;
      frmMain.conn.ExecuteDirect(QueryString);
      frmMain.Trans.Commit;


      QueryString:='CREATE UNIQUE INDEX CountersIndex ' +
                        'ON Counters(id ASC)';
      frmMain.Trans.StartTransaction;
      frmMain.conn.ExecuteDirect(QueryString);
      frmMain.Trans.Commit;


      QueryString:='INSERT INTO Counters VALUES(1,0)';
      frmMain.Trans.StartTransaction;
      frmMain.conn.ExecuteDirect(QueryString);
      frmMain.Trans.Commit;

      QueryString:= 'CREATE TABLE IF NOT EXISTS Contacts(' +
                           'id INTEGER,' +
                           'contactName VARCHAR(255),' +
                           'phoneNumbers VARCHAR(255),' +
                           'emailAddress VARCHAR(255))';
      frmMain.Trans.StartTransaction;
      frmMain.conn.ExecuteDirect(QueryString);
      frmMain.Trans.Commit;

      QueryString:='CREATE UNIQUE INDEX ContactsIndex' +
                          ' ON Contacts(id ASC)';
      frmMain.Trans.StartTransaction;
      frmMain.conn.ExecuteDirect(QueryString);
      frmMain.Trans.Commit;
end;
```

الآن.. ننسخ السطر التالي:

```
procedure CreateDataBase;
```

ونلصقه تحت

```
uses
  Classes, SysUtils;

procedure Createdatabase;
```

في الوحدة DataBaseWorks.

ولكي نستطيع استدعاء الإجراء CreateDataBase في الوحدة الخاصة بالفورم frmMain، نقوم بالربط بين الوحدتين في الاتجاه الآخر، بحيث يمكن الاستفادة من مكونات الوحدة DataBaseWorks في كود frmMainUnit.

نذهب إلى الوحدة frmMainUnit، وتحت implementation نكتب:

```
Uses DataBaseWorks;
```

وبالتالي يمكننا الآن استدعاء الإجراء CreateDataBase وإنشاء قاعدة البيانات.

3- برمجة الحدث FormActivate للتأكد من وجود ملف قاعدة البيانات من عدمه

نأتي الآن للتحقق فعلياً من وجود ملف قاعدة البيانات أو فقدانه. وأنسب وقت لعمل ذلك هو الحدث Activate للفورم frmMain.

- نظهر الفورم frmMain أمامنا.
- في نافذة الخصائص نختار التبويب Events.
- ننقر على الحدث OnActivate نقرتين سريعتين، فيفتح الكود الخاص بهذا الحدث بالشكل التالي:

```
procedure TfrmMain.FormActivate(Sender: TObject);
begin

end;
```

نكتب بين Begin و End الكود التالي:

```
if FileExists('contacts.db') then
  begin
       conn.DatabaseName:='contacts.db';
       conn.Transaction:=Trans;
       conn.Open;
       if conn.Connected = true then showmessage('connected');
  end
else
  begin
    // make database and tables.
    Createdatabase;
    showmessage('Database was created.');
  end;
```

وفي هذا الكود:

- استخدمنا الدالة FileExists والتي ترجع القيمة True إذا كان اسم الملف المعطى موجوداً، وFalse إذا لم يكن موجوداً حتى نتأكد من وجود ملف قاعدة البيانات من عدمه.
- في حال وجود ملف قاعدة البيانات، نتصل به عن طريق Conn، وفي حالة تحقق الاتصال تظهر الرسالة Connected.
- أما في حالة عدم وجود ملف قاعدة البيانات، فيتم استدعاء الإجراء Createdatabase التابع للوحدة DataBaseWorks، الذي سينشئ قاعدة البيانات وجداولها، ثم إظهار الرسالة: Database was created.

تنسيق عرض السجلات

سنستخدم العنصر TDBGrid والذي أسميناه dgvContacts لعرض السجلات المسترجعة من الجدول Contacts. ولكن قبل استخدامه، يجب أن نقوم بتنسيقه وتهيئته قبل عرض البيانات عليه، حتى لا تظهر أسماء الحقول كما هي مخزنة في قاعدة البيانات.

ننشئ وحدة جديدة ونخزنها باسم MainWindowSetup، ونربطها مع الوحدة الخاصة بالفورم الرئيسي وهي frmMainUnit.

في الوحدة الجديدة، سننشئ إجراءين:

- الأول لتنسيق الـdgvContcts.
- والثاني لعرض كافة السجلات المخزنة في الجدول Contacts. بحيث نستدعيه عند الضرورة لاحقاً.

الإجراء Get_Grid_Ready

وفي هذا الإجراء نكتب:

```
procedure Get_Grid_Ready;
begin
  frmMain.dgvContacts.BiDiMode:=bdRightToLeft;

  frmMain.dgvContacts.Columns[0].Title.Caption:='ر.م';
  frmMain.dgvContacts.Columns[0].Width:=50;
  frmMain.dgvContacts.Columns[0].Alignment:=taCenter;
  frmMain.dgvContacts.Columns[0].Title.Alignment:=taCenter;

  frmMain.dgvContacts.Columns[1].Title.Caption:='اسم جهة الاتصال';
  frmMain.dgvContacts.Columns[1].Width:=230;
  frmMain.dgvContacts.Columns[1].Title.Alignment:=taCenter;

  frmMain.dgvContacts.Columns[2].Title.Caption:='أرقام الهواتف';
  frmMain.dgvContacts.Columns[2].Width:=250;
  frmMain.dgvContacts.Columns[2].Title.Alignment:=taCenter;
  frmMain.dgvContacts.Columns[2].Alignment:=taCenter;

  frmMain.dgvContacts.Columns[3].Title.Caption:='البريد الإلكتروني';
  frmMain.dgvContacts.Columns[3].Width:=250;
  frmMain.dgvContacts.Columns[3].Title.Alignment:=taCenter;
  frmMain.dgvContacts.Columns[3].Alignment:=taCenter;
end;
```

وفي هذا الكود:

- تم تغيير اتجاه الكتابة على dgvContacts ليكون من اليمين إلى اليسار.
- ضبط النص الظاهر أعلى العمود الأول (الترقيم يبدأ من الصفر) بـ ر.م.
- ضبط اتساع العمود الأول ليساوي 50.
- ضبط محاذاة عنوان العمود الأول في الوسط.
- ضبط محاذاة محتويات العمود الأول في الوسط.
- وهكذا لبقية الأعمدة.

الإجراء Update_Grid

وفي هذا الإجراء نكتب:

```
procedure Update_Grid;
begin
  frmMain.UpdateQuery.Close;
  frmMain.UpdateQuery.DataBase:=frmMain.conn;
  frmMain.UpdateQuery.SQL.Text:='SELECT * FROM Contacts';
  frmMain.UpdateQuery.open;

  frmMain.UpdateDataSource.DataSet:=frmMain.UpdateQuery;

  frmMain.dgvContacts.DataSource:=frmMain.UpdateDataSource;
  Get_Grid_Ready;
end;
```

وفيه:

- تم الاتصال بقاعدة البيانات. والربط بين UpdateQuery وConn.
- تم استرجاع كافة السجلات المخزنة في الجدول Contacts عن طريق UpdateQuery.
- تم استقبال الـDataset المسترجعة من UpdateQuery وحفظها في UpdateDataSource.
- تم الربط بين dgvContacts وUpdateDataSource.
- تم استدعاء الإجراء السابق Get_Grid_Ready لتنسيق الـdgvContacts، ومن ثم عرض السجلات بالتنسيق الجديد.

لعرض كافة السجلات المخزنة في الجدول Contacts، نستدعي الإجراء Update_Grid في الوقت والمكان المناسبين.

وبالتالي، نعود لبرمجة الحدث FormActivate، ونعدله بحيث يتم عرض السجلات في حالة وجود ملف قاعدة البيانات، ونحذف الرسالة الخاصة بنجاح الاتصال وبالصورة التالية:

```
procedure TfrmMain.FormActivate(Sender: TObject);
begin
  if fileexists('contacts.db') then
  begin
       conn.DatabaseName:='contacts.db';
       conn.Transaction:=Trans;
       conn.Open;
       Update_Grid;
  end
  else
  begin
    // make database and tables.
    Createdatabase;
    showmessage('Database was created.');
  end;
end;
```

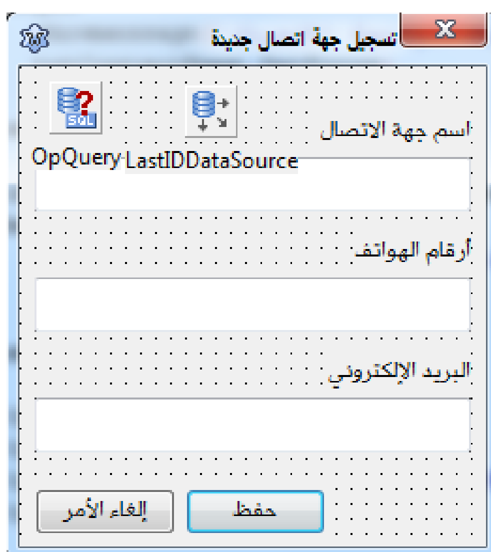
إضافة السجلات

قمنا فيما سبق بإنشاء قاعدة البيانات وجداولها، والاتصال بها، وعرض السجلات على الـdgvContacts، بالطبع لن يكون هناك سجلات، لأننا لم نضف أياً منها بعد.

لإضافة سجلات جديدة، نقوم بإنشاء نافذة جديدة، ونسميها frmNewContact، والوحدة الخاصة بها نسميها frmNewContactUnit ونضع عليها العناصر التالية:

العنصر	الخاصية	القيمة
TEdit	Name	txtContactName
TEdit	Name	txtPhoneNumbers
TEdit	Name	txtEmailAddress
TButton	Name	btnSave
	Caption	حفظ
TButton	Name	btnCancel
	Caption	إلغاء الأمر
TSQLQuery	Name	OpQuery
TDataSource	Name	LastIDDataSource

وننسقها بحيث نحصل على الصورة التالية:



نربط بين هذه الوحدة وبين كل من الوحدتين frmMainUnit وMainWindowSetup بالصورة التالية:

في الوحدة frmNewContact:

```
implementation
 uses frmMainUnit, MainWindowSetup;
```

في الوحدة frmMainUnit:

```
implementation
uses DatabaseWorks, MainWindowSetup, frmNewContactUnit;
```

ننقر الزر btnCancel نقرتين سريعتين ونكتب في حدث النقر الكود التالي:

```
procedure TfrmNewContact.btnCancelClick(Sender: TObject);
 begin
   close;
 end;
```

وهذا الكود خاص بإغلاق النافذة وإلغاء عملية الإضافة.

نقوم بتعريف متغير محلي من النوع Integer ليمثل رقم آخر سجل تمت إضافته مسبقاً في الجدول Contacts، تحت القسم Private في الوحدة frmNewContactUnit بالصورة التالية:

```
private
    { private declarations }
    LastID: Integer;
```

لنفترض أن الحقول الإجبارية هي حقل اسم جهة الاتصال وحقل أرقام الهواتف، بينما حقل البريد الإلكتروني اختياري.

الآن .. ننقر الزر btnSave ونكتب في حدث النقر الخاص به الكود التالي:

```
if txtContactName.Text = '' then
begin
     showmessage('يجب كتابة اسم جهة الاتصال');
     txtContactName.SetFocus;
     exit;
end;
```

وفي هذا الكود، نفحص المدخلات في العنصر txtContactName، فإن كانت فراغاً يتم إظهار رسالة تفيد بذلك، ثم وضع التركيز Focus فيه، والخروج من الإجراء الحالي. أي أنه عند تحقق الشرط في جملة if، لن يتم تنفيذ ما بعدها من أكواد بسبب تنفيذ الإجراء Exit.

بعد ذلك نكتب الكود الخاص بفحص العنصر txtPhoneNumbers وبالصورة التالية:

```
if txtPhoneNumbers.Text = '' then
begin
     showmessage('يجب كتابة أرقام الهواتف');
     txtPhoneNumbers.SetFocus;
     exit;
end;
```

وفي الكود السابق يتم إظهار رسالة تفيد بوجوب كتابة أرقام الهواتف في حال لم يدخلها المستخدم، ثم نقل التركيز إلى العنصر txtPhoneNumbers، والخروج من الإجراء الحالي وهو إجراء حدث النقر على الزر btnSave بسبب تنفيذ الإجراء Exit.

تحديد رقم الـID للسجل الجديد

بعد ذلك، نكتب الكود التالي:

```
OpQuery.Close;
OpQuery.DataBase:= frmMain.conn;
OpQuery.SQL.Text:= 'SELECT * FROM Counters WHERE id = 1';
OpQuery.Open;
```

وفيه ربطنا بين الـOpQuery والـConn الموجود على الفورم frmMain. ثم كتبنا جملة الاستعلام والتي ترجع السجل رقم 1 في الجدول Counters.

نستقبل الـDataset المسترجعة ونحفظها في الـLastIDDataSource بالصورة التالية:

```
LastIDDataSource.DataSet:= OpQuery;
```

الآن، الـLastIDDataSource يحتوي على سجل واحد به حقلان:

- حقل id.
- وحقل ontactid.

حقل contacted يحتفظ بآخر رقم للـid الخاص بالجدول Contacts. فمثلاً، ربما يكون آخر رقم id هو (x)، نقوم بإضافة (1) إليه، ليكون رقم الـid للسجل الجديد، ونخزنه في المتغير LastID وبالصورة التالية:

```
LastID:= LastIDDataSource.DataSet.FieldByName('contactid').AsInteger + 1;
```

يعني أننا الآن نحتفظ بقية الـid للسجل الجديد في المتغير LastID.

نقوم بتحديث قيمة الحقل contacted في الجدول Counters ليحمل القيمة الجديدة:

```
OpQuery.SQL.Text:='UPDATE Counters SET contactid=:contactid WHERE id = 1';
```

لاحظ كيفية تحديد قيمة الحقل الذي نريد تعديله، نكتب اسمه ثم (=) ثم (:) ثم القيمة المخصصة له على هيئة بارامتر.

نقوم الآن بتحديد قيمة البارامتر المرسل لجملة SQL بالصورة التالية:

```
OpQuery.ParamByName('contactid').AsInteger:=LastID;
```

ثم نعطي الأمر بتنفيذ التعديل:

```
OpQuery.ExecSQL;
```

ولا ننس أن نؤكد التحديث من خلال العنصر Trans الموجود على الفورم frmMain:

```
frmMain.Trans.Commit;
```

عملية الإضافة

الآن ... تحصلنا على رقم الـid الخاص بالسجل الجديد، فلنشرع في كتابة أكواد عملية الإضافة.

تحت آخر سطر كتبناه نكتب:

```
Opquery.SQL.Text:='INSERT INTO Contacts VALUES(' +
                              ':id,' +
                              ':contactName,'+
                              ':phoneNumbers,' +
                              ':emailAddress)';
```

وهي جملة SQL خاصة بإدراج سجل جديد، وحددنا فيها بارامترات تمثل:

- رقم الـid.
- اسم جهة الاتصال.
- أرقام الهواتف.
- البريد الإلكتروني.

الآن نرسل قيم البارامترات:

```
OpQuery.ParamByName('id').AsInteger:=LastID;
OpQuery.ParamByName('contactName').AsString:=txtContactName.Text;
OpQuery.ParamByName('phoneNumbers').AsString:=txtPhoneNumbers.Text;
OpQuery.ParamByName('emailAddress').AsString:=txtEmailAddress.Text;
```

كما نرى، يجب تحديد نوع كل بارامتر بحيث يكون متوافقاً مع النوع المقابل له في الجدول.

ونعطي الأمر لتخزين السجل الجديد:

```
OpQuery.ExecSQL;
```

وككل مرة، لا ننس أن نؤكد التحديث من خلال العنصر Trans الموجود على الفورم frmMain:

```
frmMain.Trans.Commit;
```

في نهاية هذا الإجراء، نقوم بتحديث محتويات الـdgvContacts عن طريق استدعاء الإجراء:

```
Update_Grid;
```

ثم إغلاق النافذة:

```
Close;
```

وبذلك يكون الكود النهائي للزر btnSave:

```
procedure TfrmNewContact.btnSaveClick(Sender: TObject);
begin
if txtContactName.Text = '' then
begin
       showmessage('يجب كتابة اسم جهة الاتصال');
       txtContactName.SetFocus;
       exit;
end;

if txtPhoneNumbers.Text = '' then
begin
       showmessage('يجب كتابة أرقام الهواتف');
       txtPhoneNumbers.SetFocus;
       exit;
end;

// Getting last id
OpQuery.Close;
OpQuery.DataBase:= frmMain.conn;
OpQuery.SQL.Text:= 'SELECT * FROM Counters WHERE id = 1';
OpQuery.Open;

LastIDDataSource.DataSet:= OpQuery;

LastID:= LastIDDataSource.DataSet.FieldByName('contactid').AsInteger + 1;

OpQuery.SQL.Text:='UPDATE Counters SET contactid=:contactid WHERE id = 1';
OpQuery.ParamByName('contactid').AsInteger:=LastID;
OpQuery.ExecSQL;

frmMain.Trans.Commit;
// -------------- end getting last id --------------

Opquery.SQL.Text:='INSERT INTO Contacts VALUES(' +
                               ':id,' +
                               ':contactName,'+
                               ':phoneNumbers,' +
                               ':emailAddress)';
OpQuery.ParamByName('id').AsInteger:=LastID;

OpQuery.ParamByName('contactName').AsString:=txtContactName.Text;
OpQuery.ParamByName('phoneNumbers').AsString:=txtPhoneNumbers.Text;
OpQuery.ParamByName('emailAddress').AsString:=txtEmailAddress.Text;
OpQuery.ExecSQL;

frmMain.Trans.Commit;

Update_Grid;
close;
end;
```

ولكي نجهز لعملية الإضافة التالية، نمسح صناديق النصوص، ونصفر المتغير LastID ، ثم نضع التركيز في صندوق النصوص الخاص باسم جهة الاتصال، وذلك بكتابة الكود التالي في الحدث FormClose التابع للفورم frmNewContact:

```
procedure TfrmNewContact.FormClose(Sender: TObject;
  var CloseAction: TCloseAction);
begin
     txtContactName.Clear;
     txtPhoneNumbers.Clear;
     txtEmailAddress.Clear;
     LastID:=0;
     txtContactName.SetFocus;
end;
```

لو شغلنا البرنامج الآن، ونقرنا على الزر (جديد) لن يحدث أي شيء، والسبب هو أننا لم نكتب الكود الخاص بالنقر على هذا الزر.

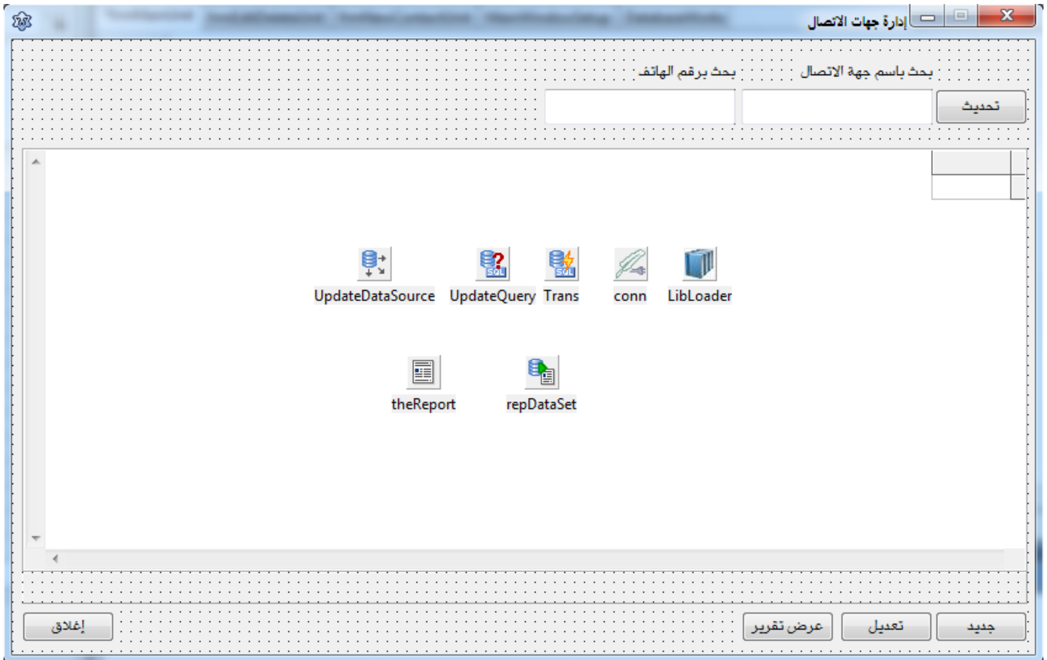
نتقل إلى الفورم frmMain، ننقر على الزر (جديد) نقرتين سريعتين ونكتب الكود التالي في حدث النقر:

```
procedure TfrmMain.btnNewClick(Sender: TObject);
begin
  frmNewContact.ShowModal;
end;
```

الآن.. نشغل، ونجرب.

البحث عن السجلات

بالنظر إلى النافذة frmMain:



نجد أن هناك:

- العنصر txtSeachInNames: للبحث باسم جهة الاتصال.
- والعنصر txtSearchInNumbrs: للبحث برقم الهاتف.

والفكرة هي السماح للمستخدم بالبحث عن اسم جهة الاتصال مثلاً بمجرد إدخال أول حرف، فيتم فلترة السجلات التي يوجد فيها ذلك الحرف، ثم إعادة الفلترة عند إدخال الحرف الثاني، والثالث وهكذا، لتضييق دائرة البحث، وعرض السجلات المفلترة لحظياً على العنصر dgvContacts.

وفي حال لا يوجد تطابق بين الحروف المدخلة ونظيرتها في قواعد البيانات، لا يتم عرض أي شيء على الـdgvContacts.

يعني تتم عمليةالفلترة وعرض المقترحات في كل مرة يدخل المستخدم فيها حرفاً، أو رقماً.

تماماً مثل ما نبحث عن جهات الاتصال في جهاز الهاتف.

البحث باسم جهة الاتصال

ننقر على صندوق النصوص txtSearchInNames نقرتين سريعتين، فيظهر الكود الخاص بالحدث Change الخاص بهذا العنصر:

```
procedure TfrmMain.txtSearchInNamesChange(Sender: TObject);
begin

end;
```

في كل مرة يقوم المستخدم بإدخال حرف، نقوم بالبحث عن ما يشابه المكتوب في txtSearchInNames.Text في سجلات الجدول Contacts، وعرض النتيجة على الـdgvContacts. أما إن لم نجد تشابهاً أو تطابقاً، فلا يتم عرض أي شيء.

نكتب بين Begin و End:

```
  if txtSearchInNames.Text = '' then
  begin
       Update_Grid;
       exit;
  end;
```

وفي الكود السابق، نفحص المكتوب في العنصر txtSearchInNames، فإن كان فراغاً، نعرض كافة السجلات على dgvContacts ثم نخرج من الإجراء. أما إن لم يكن فراغاً، أي أن الـ txtSearchInNames يحتوي على حرف على الأقل، فنبحث في قاعدة البيانات عن السجلات التي يكون فيها حقل الـcontactName مشابهاً للقيمة المعطاة في txtSearchInNames:

```
UpdateQuery.Close;
UpdateQuery.SQL.Text:='SELECT * FROM Contacts ' +
                             'WHERE contactName LIKE ''%'
                             + txtSearchInNames.Text + '%''';
UpdateQuery.Open;
```

بعد ذلك، نربط بين الـ UpdateQuery وUpdateDataSource:

```
UpdateDataSource.DataSet:=UpdateQuery;
```

ثم نربط بين الـdgvContacts والـUdateDataSource:

```
dgvContacts.DataSource:=UpdateDataSource;
```

وأخيراً، نستدعي إجراء تنسيق الـdgvCntacts:

```
Get_Grid_Ready;
```

وفيما يلي الإجراء بالكامل:

```
procedure TfrmMain.txtSearchInNamesChange(Sender: TObject);
begin
  if txtSearchInNames.Text = '' then
  begin
       Update_Grid;
       exit;
  end;
  UpdateQuery.Close;
  UpdateQuery.SQL.Text:='SELECT * FROM Contacts ' +
                              'WHERE contactName LIKE ''%'
                              + txtSearchInNames.Text + '%''';
  UpdateQuery.Open;
  UpdateDataSource.DataSet:=UpdateQuery;
  dgvContacts.DataSource:=UpdateDataSource;
  Get_Grid_Ready;
end;
```

البحث برقم الهاتف

نـنقر على العنصر txtSearchInNumbers نقرتين سريعتين، ونكتب في حدث الـ Change الكود التالي:

```
procedure TfrmMain.txtSearchInNumbersChange(Sender: TObject);
begin
  if txtSearchInNumbers.Text = '' then
  begin
       Update_Grid;
       exit;
  end;
  UpdateQuery.Close;
  UpdateQuery.SQL.Text:='SELECT * FROM Contacts ' +
                              'WHERE phoneNumbers LIKE ''%'
                              + txtSearchInNumbers.Text + '%''';
  UpdateQuery.Open;
  UpdateDataSource.DataSet:=UpdateQuery;
  dgvContacts.DataSource:=UpdateDataSource;
  Get_Grid_Ready;
end;
```

وهو نفس الكود السابق، فقط غيرنا اسم صندوق النصوص، والحقل الذي نريد البحث من خلاله.

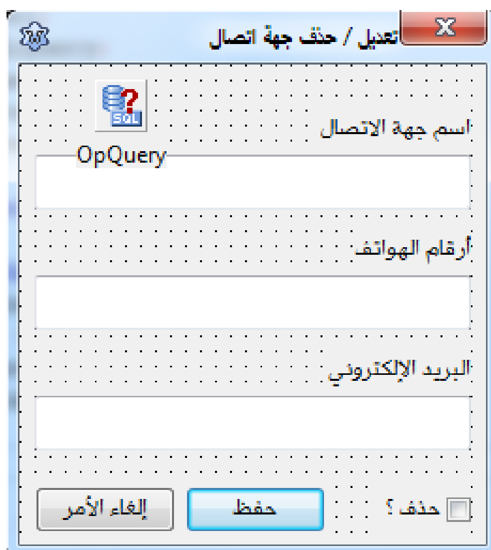
نشغل ونجرب.

تعديل / حذف السجلات

لبرمجة جزئية التعديل أو الحذف، نقوم بإنشاء نافذة جديدة، ونسميها frmEditDelete، والوحدة الخاصة بها نسميها frmEditDeleteUnit ونضع عليها العناصر التالية:

العنصر	الخاصية	القيمة
TEdit	Name	txtContactName
TEdit	Name	txtPhoneNumbers
TEdit	Name	txtEmailAddress
TButton	Name	btnSave
	Caption	حفظ
TButton	Name	btnCancel
	Caption	إلغاء الأمر
TCheckBox	Name	chkDelete
	Caption	حذف ؟
	Checked	False
TSQLQuery	Name	OpQuery

وننسقها بحيث نحصل على الصورة التالية:



نقوم بالتعريف المتبادل بين frmMainUnit وMainWindowSetup من جهة والوحدة frmEditDelete من جهة أخرى.

الفكرة كالتالي: في النافذة الرئيسية، يختار المستخدم سجلاً معيناً بالنقر المزدوج عليه، وبذلك يتم تفعيل الزر (تعديل)، وعند النقر عليه يتم إرسال بيانات السجل إلى الفورم frmEditDelete، ويتم عرضه أمام المستخدم ليتخذ قراره إما بالتعديل، أو بالحذف عن طريق التأشير على الخيار (حذف ؟) أسفل الفورم، ثم النقر على الزر (حفظ).

نبرمج الزر إلغاء الأمر ونكتب في حدث النقر الخاص به ما يلي:

```
procedure TfrmEditDelete.btnCancelClick(Sender: TObject);
begin
  close;
end;
```

الآن.. نحتاج إلى تعريف مجموعة من المتغيرات العمومية Public، فنصعد إلى بداية الوحدة frmEditDeleteUnit، وتحت القسم Public نكتب:

```
  public
    { public declarations }
    ContactID: Integer;
    ContactName: String;
    ContactPhones: String;
    ContactEmail: String;
```

حيث ContactID يمثل رقم الـ id الخاص بالسجل، ContactName يمثل اسم جهة الاتصال، ContactPhones يمثل أرقام الهواتف، وContactEmail يمثل البريد الإلكتروني.

هذه المتغيرات ستستقبل القيم المرسلة من الفورم frmMain إلى الفورم frmEditDelete.

في حدث FormActivate، نكتب الكود التالي:

```
Opquery.Close;
Opquery.DataBase:=frmMain.conn;
```

وذلك للربط بين العنصر OpQuery والـ Conn التابع للفورم frmMain.

ثم نقوم بوضع القيم المستقبلة من الفورم الرئيسي في ما يقابلها من صناديق النصوص على الفورم الحالي:

```
txtContactName.Text:=ContactName;
txtPhoneNumbers.Text:=ContactPhones;
txtEmailAddress.Text:=ContactEmail;
```

ثم وضع التركيز Focus في الصندوق الأول الخاص باسم جهة الاتصال:

```
txtContactName.SelectAll;
```

فيكون الكود كاملاً كالتالي:

```
procedure TfrmEditDelete.FormActivate(Sender: TObject);
begin
  Opquery.Close;
  Opquery.DataBase:=frmMain.conn;

  txtContactName.Text:=ContactName;
  txtPhoneNumbers.Text:=ContactPhones;
  txtEmailAddress.Text:=ContactEmail;

  txtContactName.SelectAll;
end;
```

الآن.. لدينا رقم الحقل المطلوب تعديل بياناته أو حذفه مخزناً في المتغير ContactID.

نأتي لبرمجة الزر (حفظ)، نكتب في كود حدث النقر:

```
  if txtContactName.Text = '' then
  begin
    showmessage('يجب كتابة اسم جهة الاتصال');
    txtContactName.SetFocus;
    exit;
  end;

  if txtPhoneNumbers.Text = '' then
  begin
    showmessage('يجب كتابة أرقام الهواتف');
    txtPhoneNumbers.SetFocus;
    exit;
  end;
```

وهو نفس الكود الذي استخدمناه لفحص المدخلات في عملية الإضافة.

بعد ذلك نفحص العنصر chkDelete، فإن كان مؤشراً (عليه علامة صح) نقوم بحذف السجل الحالي، وتحديث الفورم الرئيسي:

```
  if chkDelete.Checked = true then
  begin
    OpQuery.SQL.Text:='DELETE FROM Contacts WHERE id=:id';
    OpQuery.ParamByName('id').AsInteger:=ContactID;
    OpQuery.ExecSQL;
    frmMain.Trans.Commit;
    Update_Grid;
  end;
```

أما إذا كان غير مؤشر بعلامة صح، فهذا يعني أن العملية تعديل على السجل الحالي:

```
if chkDelete.Checked = false then
begin
  OpQuery.SQL.Text:='UPDATE Contacts SET ' +
                            'contactName=:contactName, ' +
                            'phoneNumbers=:phoneNumbers, ' +
                            'emailAddress=:emailAddress ' +
                            ' WHERE id=:id';
  OpQuery.ParamByName('contactName').AsString:=txtContactName.Text;
  OpQuery.ParamByName('phoneNumbers').AsString:=txtPhoneNumbers.Text;
  OpQuery.ParamByName('emailAddress').AsString:=txtEmailAddress.Text;
  OpQuery.ParamByName('id').AsInteger:=ContactID;
  OpQuery.ExecSQL;
  frmMain.Trans.Commit;
  Update_Grid;
end;
```

وأخيراً، نقوم بإغلاق الفورم الحالي:

```
close;
```

وفيما يلي الكود كاملاً:

```
procedure TfrmEditDelete.btnSaveClick(Sender: TObject);
begin
  if txtContactName.Text = '' then
  begin
    showmessage('يجب كتابة اسم جهة الاتصال');
    txtContactName.SetFocus;
    exit;
  end;
  if txtPhoneNumbers.Text = '' then
  begin
    showmessage('يجب كتابة أرقام الهواتف');
    txtPhoneNumbers.SetFocus;
    exit;
  end;
  if chkDelete.Checked = true then
  begin
    OpQuery.SQL.Text:='DELETE FROM Contacts WHERE id=:id';
    OpQuery.ParamByName('id').AsInteger:=ContactID;
    OpQuery.ExecSQL;
    frmMain.Trans.Commit;
    Update_Grid;
  end;
  if chkDelete.Checked = false then
  begin
    OpQuery.SQL.Text:='UPDATE Contacts SET ' +
                            'contactName=:contactName, ' +
                            'phoneNumbers=:phoneNumbers, ' +
                            'emailAddress=:emailAddress ' +
                            ' WHERE id=:id';
    OpQuery.ParamByName('contactName').AsString:=txtContactName.Text;
    OpQuery.ParamByName('phoneNumbers').AsString:=txtPhoneNumbers.Text;
    OpQuery.ParamByName('emailAddress').AsString:=txtEmailAddress.Text;
    OpQuery.ParamByName('id').AsInteger:=ContactID;
    OpQuery.ExecSQL;
    frmMain.Trans.Commit;
    Update_Grid;
  end;
  close;
end;
```

وللتمهيد لعملية التعديل / الحذف التالية، نكتب الكود التالي في حدث الإغلاق للفورم frmEditDelete:

```
procedure TfrmEditDelete.FormClose(Sender: TObject;
  var CloseAction: TCloseAction);
begin
  txtContactName.Clear;
  txtPhoneNumbers.Clear;
  txtEmailAddress.Clear;

  ContactID:=0;
  ContactName:='';
  ContactPhones:='';
  ContactEmail:='';
  chkDelete.Checked:=False;
end;
```

كان هذا ما يتعلق ببرمجة الفورم frmEditDelete، والآن نقوم ببرمجة الفورم frmMain للتجهيز لعملية التعديل / الحذف.

في البداية، نعرف نفس المتغيرات التي عرفناها على الفورم frmEditDelete هنا في الفورم الرئيسي، تحت القسم Private وكما يلي:

```
  private
    { private declarations }
    ContactID: Integer;
    ContactName: String;
    ContactPhones: String;
    ContactEmail: String;
```

نبرمج حدث النقر المزدوج للـdgvContacts، نختارها في وضع التصميم، وننتقل إلى نافذة الخصائص، في التبويب Events نختار حدث النقر المزدوج، فنكتب الكود التالي:

```
procedure TfrmMain.dgvContactsDblClick(Sender: TObject);
begin
  if dgvContacts.DataSource.DataSet.FieldByName('id').AsInteger > 0 then
     begin
        pnlSelectedContact.Caption:=
        dgvContacts.DataSource.DataSet.FieldByName('contactName').AsString;

        ContactID:=
        dgvContacts.DataSource.DataSet.FieldByName('id').AsInteger;

        ContactName:=
        dgvContacts.DataSource.DataSet.FieldByName('contactName').AsString;

        ContactPhones:=
        dgvContacts.DataSource.DataSet.FieldByName('phoneNumbers').AsString;

        ContactEmail:=
        dgvContacts.DataSource.DataSet.FieldByName('emailAddress').AsString;
        btnEdit.Enabled:=true;
     end
```

```
  else begin
    pnlSelectedContact.Caption:='';
    ContactID:= 0;
    ContactName:= '';
    ContactPhones:= '';
    ContactEmail:= '';
    btnEdit.Enabled:=false;
  end;
end;
```

وفي الكود السابق:

- تم فحص قيمة حقل الـid للصف المحدد في الـdgvContacts، فإن كانت قيمته أكبر من الصفر، يعني تم اختيار سجل حقيقي، وبالتالي نقوم بتخصيص قيم الحقول إلى ما يقابلها من متغيرات.
- طباعة اسم جهة الاتصال على العنصر pnlSelectedContact كي يراه المستخدم ويتأكد بأنه اختار السجل المطلوب.
- تفعيل الزر (تعديل) لإتاحة الفرصة للمستخدم بالقيام بعملية التعديل أو الحذف.
- أما في حالة عدم اختيار سجل، فيتم تصفير رقم الـid ومسح قيم بقية المتغيرات، ثم إبطال عمل الزر (تعديل).

نأتي الآن لبرمجة الزر (تعديل) والذي يعمل فقط في حال اختيار سجل من الـdgvContacts:

```
procedure TfrmMain.btnEditClick(Sender: TObject);
begin
    frmEditDelete.ContactID:=ContactID;
    frmEditDelete.ContactName:=ContactName;
    frmEditDelete.ContactPhones:=ContactPhones;
    frmEditDelete.ContactEmail:=ContactEmail;
    frmEditDelete.ShowModal;

    pnlSelectedContact.Caption:='';
    ContactID:= 0;
    ContactName:= '';
    ContactPhones:= '';
    ContactEmail:= '';
    btnEdit.Enabled:=false;
end;
```

حيث تم:

- إرسال قيمة المتغير ContactID الخاص بالفورم frmEditDelete.
- إرسال قيمة المتغير ContactName الخاص بالفورم frmEditDelete.
- إرسال قيمة المتغير ContactPhones الخاص بالفورم frmEditDelete.
- إرسال قيمة المتغير ContactEmail الخاص بالفورم frmEditDelete.
- ثم إظهار الفورم frmEditDelete.

- وبعد القيام بما هو مطلوب وإغلاق الفورم frmEditDelete، نقوم بتصفير المتغيرات على الفورم frmMain.

زر التحديث

وظيفة هذا الزر هو إلغاء عملية اختيار سجل معين، وتصفير المتغيرات، ومسح صناديق نصوص البحث، وإبطال عمل الزر (تعديل)، وأخيراً عرض سجلات الجدول Contacts:

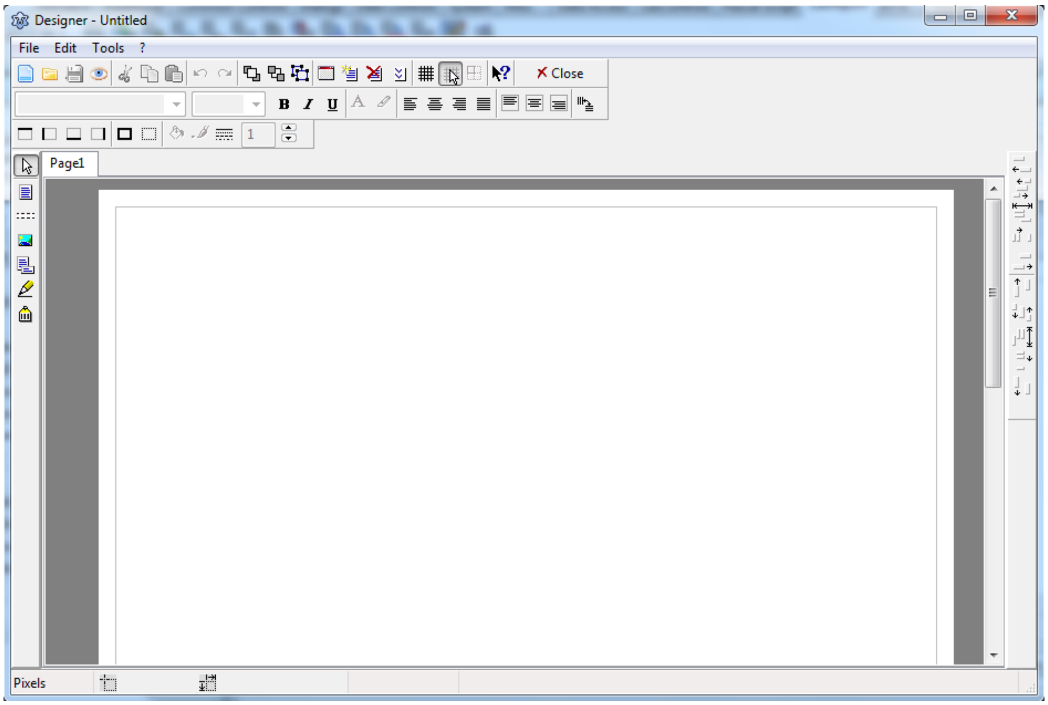
```
procedure TfrmMain.btnRefreshClick(Sender: TObject);
begin
     txtSearchInNames.Clear;
     txtSearchInNumbers.Clear;
     pnlSelectedContact.Caption:='';
     ContactID:= 0;
     ContactName:= '';
     ContactPhones:= '';
     ContactEmail:= '';
     btnEdit.Enabled:=false;
     Update_Grid;
end;
```

نشغل ونجرب.

إنشاء التقرير

سأفترض– عزيزي – أن وحدة تقارير LazReports مثبتة مع نسخة لازاروس خاصتك.

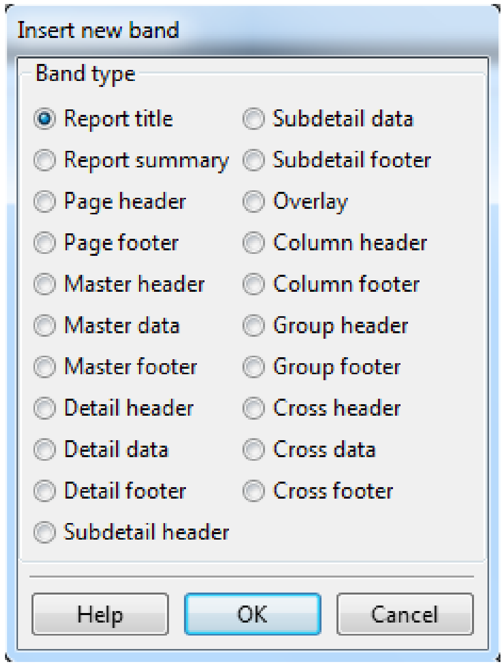
على الفورم الرئيسي frmMain، ننقر فوق العنصر TheReport نقرتين سريعتين (أو نقرة يمين واختيار Design report) لفتح مصمم التقارير والذي سيكون بالصورة التالية:



تتكون نافذة مصمم التقارير من شريط القوائم، وأشرطة الأدوات الأفقية للتحرير، وشريط جانبي لمكونات التقرير.

قبل البدء في تصميم التقرير يتوجب تعديل خصائص صفحة التقرير، وتحديد حجم الورق، من خلال فتح القائمة File ثم Page Options.

يتم تقسيم صفحة التقرير إلى Bands، وكل Band يختص بجزئية معينة.



فـ Report title مثلاً يختص بعنوان التقرير، بينما Page Header لرأس الصفحة.. وهكذا.

ولكن أهم الـBands التي نستعملها هي:

نوع الـBand	الوصف
Page Header	لرأس الصفحة
Master Data	معلومات رأس الفاتورة مثلاً، يمكن جلبها من قاعدة البيانات.
Detail Header	المكان المناسب لتصميم رأس جدول الفاتورة مثلاً.
Detail Data	بيانات الجدول المكررة، يمكن جلبها من قاعدة البيانات.
Detail Footer	إحصائيات تحت الجدول غير مكررة كالمبلغ الإجمالي تحت الفاتورة مثلاً، يمكن جلبها من قاعدة البيانات
Page Footer	تذييل الصفحة

لإدراج حقل في أي Band نستخدم العنصر rectangle وبداخله نكتب اسم الحقل بين قوسين مربعين.

مثلاً.

```
[contactName]
```

البدء في العمل

في حدث النقر على الزر (عرض التقرير) نكتب:

```
UpdateQuery.Close;
UpdateQuery.DataBase:=conn;
UpdateQuery.SQL.Text:='SELECT * FROM Contacts';
UpdateQuery.open;

UpdateDataSource.DataSet:=UpdateQuery;
```

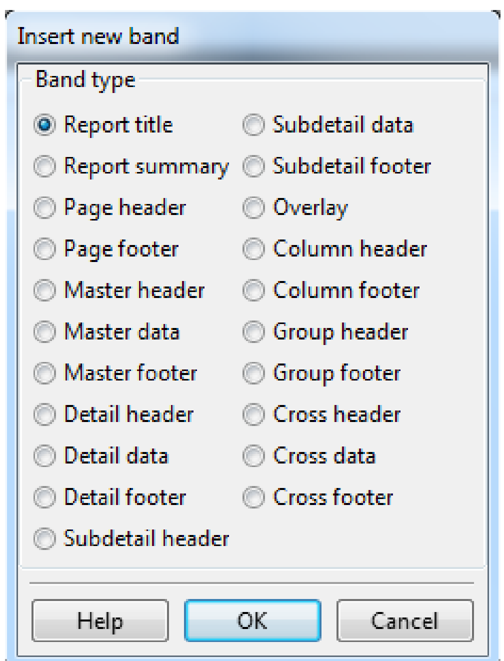
وهذا كود جلب السجلات من الجدول Contacts ووضع الـDataset المسترجعة في الـUpdateDataSource.

بعد ذلك نربط بين الـRepDataSet وهو Dataset خاصة بالتقارير، مع الـUpdateDataSource كما يلي:

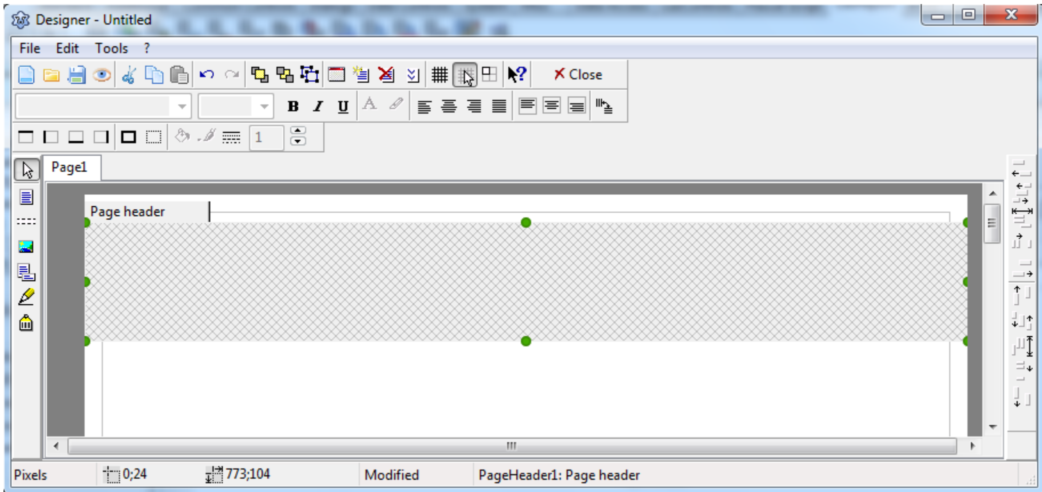
```
RepDataset.DataSource:=UpdateDataSource;
```

تبقى جزئية عرض التقرير، لكننا سنرجع ونكتب كودها بعد تجهيز التقرير أولاً.

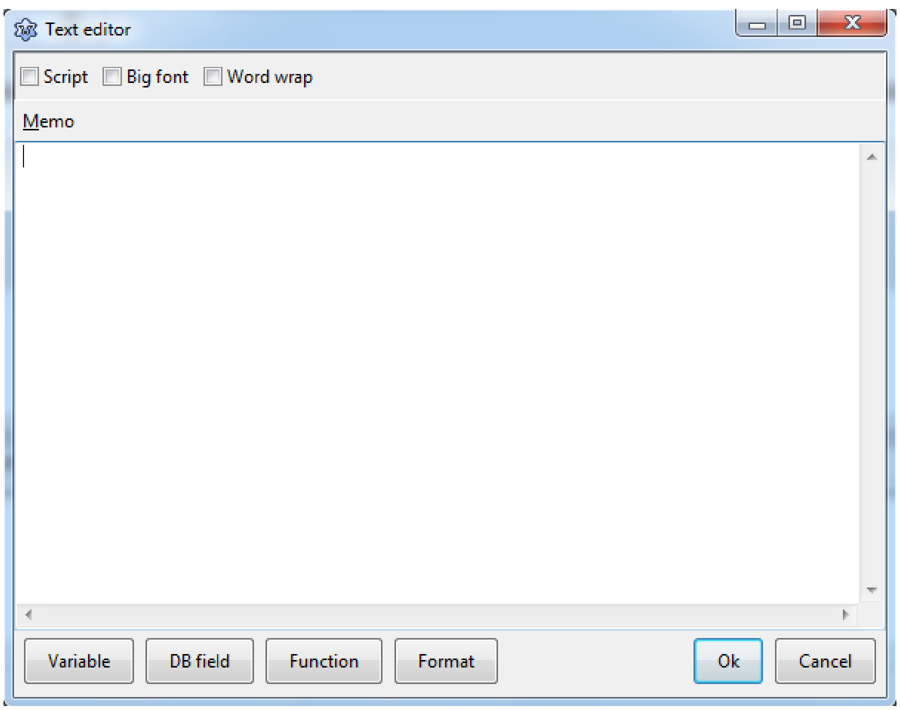
نختار الـBand من شريط الأدوات العمودي، ونضعه على الصفحة، فتظهر نافذة تحديد الـBand المطلوب:



فنختار Page Header. فيتم وضع الـBand على الصفحة كما بالصورة التالية:



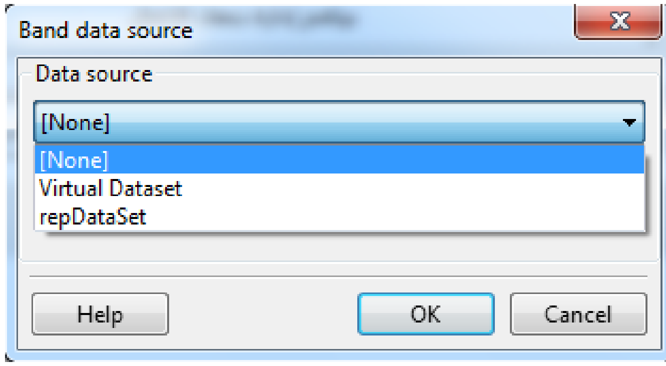
الآن نضع مربع نصي على هذا الباند، فتظهر نافذة للكتابة فيه كما الصورة التالية:



فنكتب فيها مثلاً: برنامج إدارة جهات الاتصال، ثم ننقر الزر Ok. نعدل نوع وحجم خط المربع النصي حتى يظهر بشكل مناسب.

نخزن التقرير باسم ContactsReport.lrf في نفس مجلد المشروع.

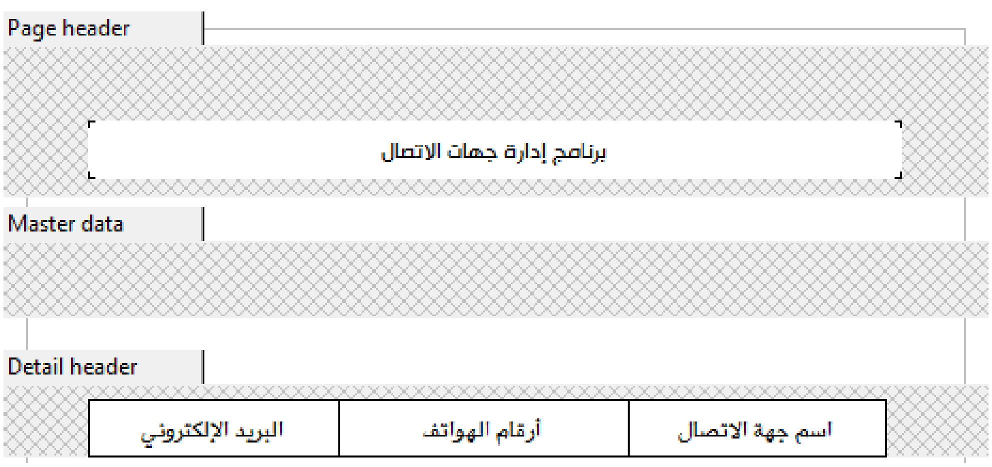
نعود لمصمم التقرير، نضع باند جديد من النوع Master Data، فتظهر نافذة لاختيار مصدر البيانات التي سنعرضها في هذا الباند، كما الصورة التالية:



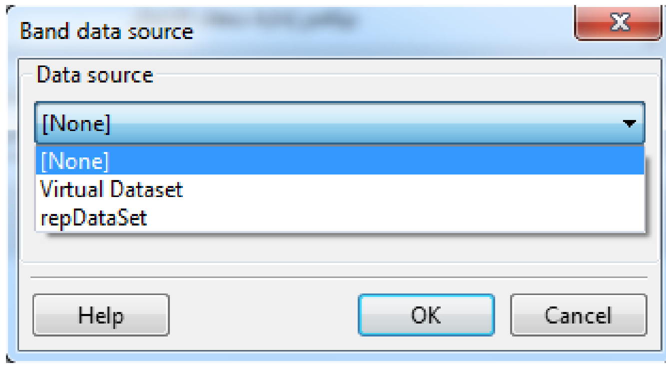
فنختار الأخيرة repDataSet ثم OK.

لن نعرض أي شيء على هذا الباند، ولكن وجوده ضروري حتى تظهر بقية أجزاء التقرير عند التنفيذ.

نضع باند جديد من نوع Detail Header على صفحة التقرير، ننقر عليه يمين ثم نختار Show on all pages. نضع ثلاثة مربعات نصية وننسقها كما بالصورة التالية:

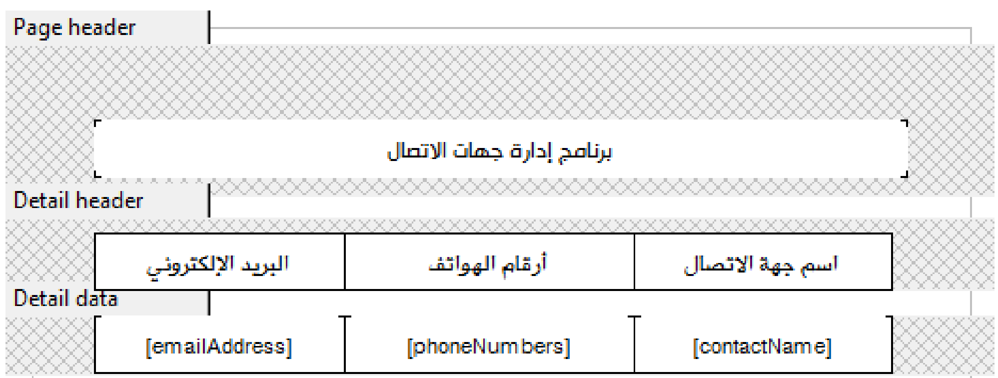


نضع باند جديد من النوع Detail Data، فتظهر نافذة لاختيار مصدر البيانات التي سنعرضها في هذا الباند، كما الصورة التالية:



فنختار الأخيرة repDataSet ثم OK.

نضع ثلاثة مربعات نصوص، وننسقها بحيث تظهر كما في الصورة التالية:



نحفظ التقرير، ونغلق مصمم التقارير.

نعود لبرمجة الزر (عرض التقرير)، وتحت آخر سطر فيه نكتب:

```
TheReport.LoadFromFile('ContactsReport.lrf');
TheReport.ShowReport;
```

السطر الأول لتحديد اسم ملف التقرير، والثاني للبدء في عرض التقرير أمام المستخدم.

بعد ذلك، نقوم بتحديث البيانات على الفورم الرئيسي:

```
Update_Grid;
```

وهذا هو الكود كاملاً:

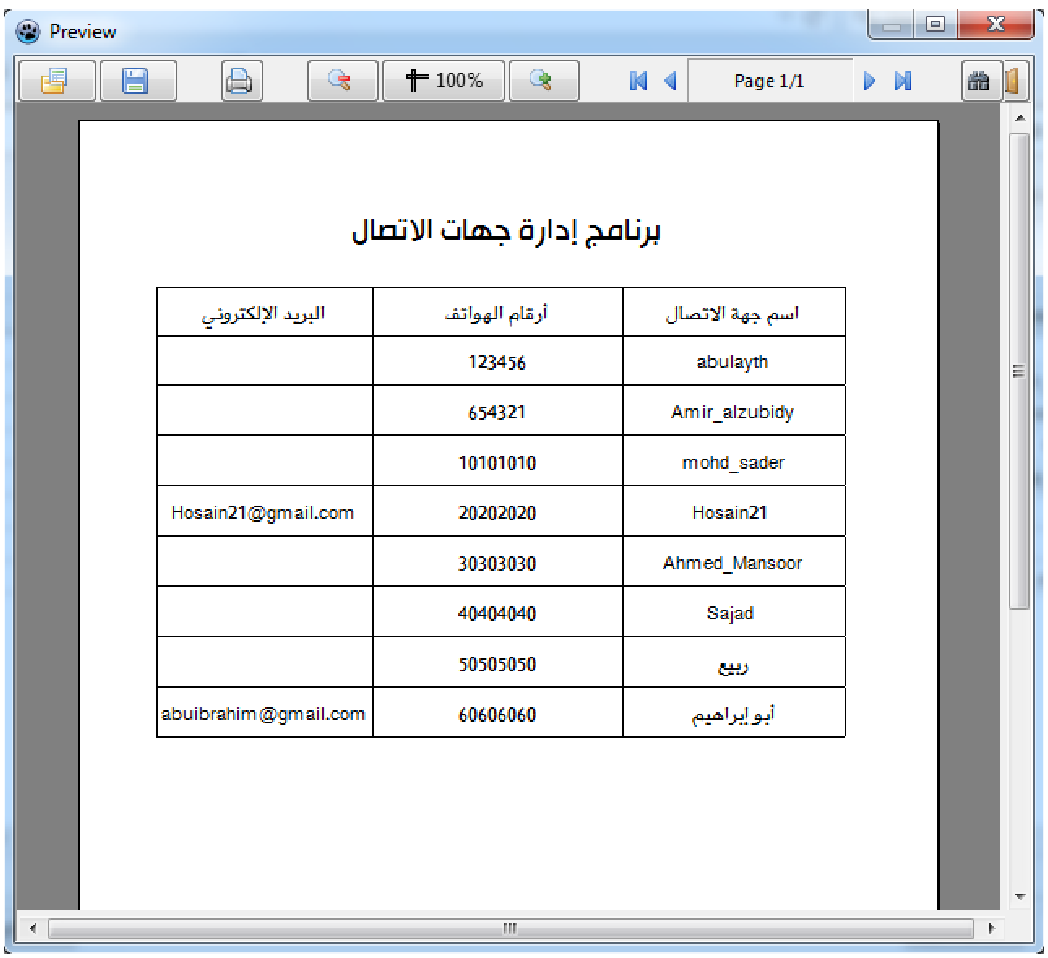
```
procedure TfrmMain.btnShowReportClick(Sender: TObject);
begin
  UpdateQuery.Close;
  UpdateQuery.DataBase:=conn;
  UpdateQuery.SQL.Text:='SELECT * FROM Contacts';
  UpdateQuery.open;

  UpdateDataSource.DataSet:=UpdateQuery;

  RepDataset.DataSource:=UpdateDataSource;

  TheReport.LoadFromFile('ContactsReport.lrf');
  TheReport.ShowReport;
  Update_Grid;
end;
```

وهذه لقطة من التقرير أثناء التشغيل:



برنامج إدارة جهات الاتصال

البريد الإلكتروني	أرقام الهواتف	اسم جهة الاتصال
	123456	abulayth
	654321	Amir_alzubidy
	10101010	mohd_sader
Hosain21@gmail.com	20202020	Hosain21
	30303030	Ahmed_Mansoor
	40404040	Sajad
	50505050	ربيع
abuibrahim@gmail.com	60606060	أبو إبراهيم